Notes on *Engineering a Sort Function*[*]

Sudarshan S. Chawathe

THESE NOTES, in the guise of questions, are meant to be read in conjunction with

> Jon L. Bentley and M. Douglas McIlroy. Engineering a sort function. *Software–Practice and Experience*, 23(11):1249–1265, November 1993.

1. Give an example of a stable sorting method and explain why it is stable. Give an example of a non-stable sorting method, other than the method in this paper, and explain why it is not stable.

2. Justify the following claim made by the authors on page 1250, just below the `iisort` listing: "Insertion sort uses about $n^2/4$ comparisons on a randomly permuted array, and it never uses more than $n^2/2$ comparisons."

3. Refer to Program 1 on page 1251. What is the range of values of the parameter `n` for which the `swap` function behaves correctly?

4. On page 1251, just below Program 1, why are the strings required to be null-terminated even though they are all `len` bytes long?

5. Justify the following claim about Program 2, made near the bottom of page 1251: "On arrays that are already sorted, it makes roughly $n^2/2$ comparisons."

6. Prove the correctness of `iqsort0` on page 1252. [Hint: Introduce and prove some key invariants, and show that the invariants imply correctness.]

7. The description of the method to partition an array using two indices (middle of page 1252) notes that it is important that both indices stop on elements equal to the pivot, else the running time is quadratic for an input array composed of a random distribution of zeros and ones. Justify this claim.

8. Generate a table analogous to Table I (page 1254) for a programming environment of your choice. The programming environment consists of the hardware, operating system, programming language, compiler or interpreter, and any associated libraries. Describe the environment and the method used to generate your table in detail sufficient to enable others to reproduce your results. Comment on how well your results are suited to an analysis of the kind found in the paper.

9. Justify the claim (bottom of page 1254) that partitioning about a random element takes roughly $1.386 \lg n$ comparisons.

10. Justify the claim (page 1255, just below Program 5) that Program 5 finds the median of three elements using 8/3 comparisons on average.

11. Does the code near the bottom of page 1255 work properly when `n` is not divisible by 2? (Note the use of integer division on the first and sixth lines.) Explain your answer using a suitable example.

12. Near the bottom of page 1255 is the comment "We could get fancier and randomize the sample," followed by "but a library sort has no business side-effecting the random number generator." Explain both quoted comments. What is the benefit of randomizing the sample? Why is the stated side effect undesirable? Is there a way to use randomization without this side effect?

13. The paper notes that the Seventh Edition `qsort` finishes after a single scan when given an input of equal elements (top of page 1257). Use a suitable example to explain this behavior.

14. Provide a formal definition of the Dutch National Flag problem mentioned on page 1257.

15. Describe the precise invariants suggested by the second, third, and fourth diagrams on page 1257,

[*]Class notes for the Computer Science Capstone course at the University of Maine, Spring 2008, for 3rd-year undergraduate students. `chaw@cs.umaine.edu`.

and explain the resulting solutions to the Dutch National Flag problem using suitable examples.

16. On page 1258 is the comment "The disorder induced by swapping the partition element to the beginning is costly when the input is ordered in reverse or near-reverse." Justify this comment and explain your answer using a suitable example.

17. Item 4 at the top of page 1259 notes "We guard recursive calls to Quicksort on `n` elements with the test `if (n > 1)`." What is the motivation for this modification?

18. Prove that "by sorting the larger side of the partition last and eliminating tail recursion, we could reduce worst-case stack growth from linear in n to logarithmic" (middle of page 1259).

19. Provide a modified version of the `qsort` function (Program 7) that incorporates the suggestion of Question 18.

20. Refer to the five observations on "well known roads to optimization that we have not traveled" (near the bottom of page 1259). Comment on their applicability to the environment used for Question 8.

21. Provide a method that, given a positive integer n as input, produces as output an array $A(n)$ of n integers such that the running time of `qsort` (Program 7) with input $A(n)$ (and the usual comparison function for integers) is quadratic in n. [Hint: See the footnote on page 1261.]

22. Write a variant of Program 7 by using the suggestions near the bottom of page 1262.

23. Comment on how the "lessons that apply well beyond sorting" (page 1263) apply to your recent or ongoing work, such as your Capstone project.

24. Generate a table analogous to Table IV (page 1264) for two or more current architectures. Describe how your results may be reproduced.

25. The poor performance of char-wise swapping on the MIPS R3000 architecture is attributed to cache interference (bottom of page 1262). Expand on this explanation and comment on how it applies to current architectures.

26. Explain how the definition of the `SWAPINIT` macro (page 1264) sets `swaptype` to the correct value, as defined in the text preceding the macro definition.